Numerical Methods ITGS219

Lecture 2: Writing Scripts and Functions

By: Zahra Abdalla Elashaal

Course Contents:

Creating Scripts and Functions

A script is simply a file containing the sequence of MATLAB commands which we wish to execute to solve the task at hand; in other words a script is a computer program written in the language of Matlab.

- To invoke the MATLAB editor we type *edit* at the prompt. This editor has the advantage of understanding MATLAB syntax and producing automatic formatting.

Example: We begin by entering and running the code:

```
a = input ('First number ');
b = input ('Second number ');
disp([' Their sum is ' num2str (a+b)])
disp([' Their product is ' num2str (a*b)])
```

We shall create our first script and save it in a file named *twonums.m* .

MATLAB will have given this code the default name *Untitled.m*.

To execute the file *twonums* enter its name at the prompt of the command window

contents of the file can be displayed by typing *type twonums*.

Example has three new commands, input , num2str and disp

The input command prompts the user with the flag contained within the quotes ' ' and takes the user's response from the standard input,

The second command num2str stands for number-to-string

This is then displayed using the disp.

Creating Scripts and Functions

Use either the command *help* or the command *which* in combination with the filename, for instance *help load* or *which load* for the Matlab command load.

to check that you are in the correct directory use the command *pwd* to 'print working directory'

You can also list the files in the current directory by typing *dir* or alternatively all the available MATLAB files can be listed by using *what*: for more details see *help what*.

Example: If we create a MATLAB file called *power.m* using the editor it can be saved in the current directory: however the code will not work. The reason for this can be seen by typing *which power* which produces the output

>> which power
power is a built-in function.
So MATLAB will try to run the built-in function.

Important Point

- It is very important you give your files a meaningful name and that the files end with .m.
- You should avoid using filenames which are the same as the variables you are using and which coincide with Matlab commands.
- Make sure you do not use a dot in the body of the filename and that it does not start with a special character or a number.

Functions

Functions are codes take inputs and return outputs.

As the next example and we will save as *xsq.m*.

```
function [output] = xsq(input)
output = input .^ 2;
```

The variables *input* and *output* are local variables that are used by the function; they are not accessible to the general Matlab workspace.

- The first line of *xsq.m* tells us this is a function called *xsq* which takes an input called *input* and returns a value called *output*. The input is contained in round brackets, and the output is contained within square brackets. It is crucial for good practice that the name of the function *xsq* corresponds to the name of the file *xsq.m* (without the .m extension).
- The second line is to calculate the square of the value of the input, and storing this result in the variable output. Notice that the function uses dot arithmetic .^ so that this function will work with both vector and matrix inputs (performing the operation element by element).

```
>> x = 1:10;
>> y = xsq(x)
```

```
>> A = [1 2 3 4 5 6];
>> y = xsq(A)
y = 1 4 9 16 25 36
```

You can know about variables by typing:
who, which lists all variables in use, or
whos, which lists all variables along with size and type.

Functions

Example Suppose we want to plot contours of a function of two variables $z = x^2 + y^2$. We can use the code

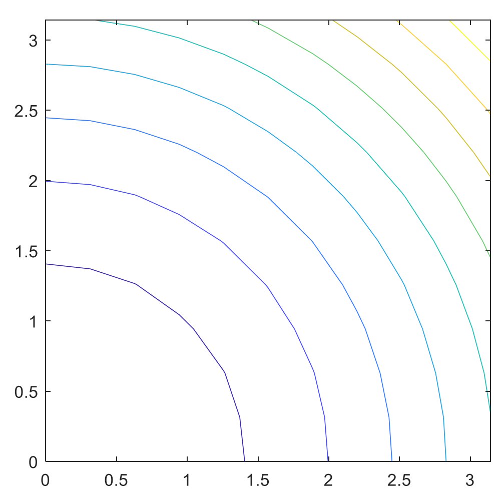
```
function [output] = func (x, y)
output = x.^2 + y.^2;
```

should be saved in the file *func.m*.

the vectors x and y must have the same size

To plot the contours (that is the level curves) of the function

```
x = 0.0:pi/10:pi;
y = x;
[X,Y] = meshgrid(x,y);
f = func(X,Y);
contour(X,Y,f)
axis([0 pi 0 pi])
axis equal
```



Functions

Example Suppose we now want to construct the squares and cubes of the elements of a vector.

```
function [sq, cub] = xpowers (input)
sq = input.^2;
cub = input.^3;
```

This function file must be saved as *xpowers.m* and it can be called as follows:

```
x = 1:10;
[xsq, xcub] = xpowers (x);
```

Notice that: when the function is called we must know what form of output we expect, whether it be a scalar, a vector or a matrix. The expected outputs should be placed within square brackets.

Example a function can have multiple inputs and outputs:

```
function [out1,out2] = multi(in1,in2,in3)
out1 = in1 + max(in2,in3);
out2 = (in1 + in2 + in3)/3;
```

```
x1 = 2; x2 = 3; x3 = 5;
[y1, y2] = multi(x1,x2,x3);
y1, y2
```

For this example we obtain y1=7 and y2=3.3333.

Example Consider a code which returns a scalar result from a vector input.

```
function [output] = sumsq(x)
output = sum(x.^2);
```

```
x = [1 2 4 5 6];
y = sumsq(x)
```

sets y equal to the scalar $1^2 + 2^2 + 4^2 + 5^2 + 6^2 = 82$.

Brief Aside

Work with matrices in the previous example will also work with matrices:

```
>> A=[1 2 3; 4 5 6];
>> sumsq(A)
    ans = 17  29  45
```

This has exploited the property that the sum command sums the columns of a matrix. If we want to sum the rows of a matrix we use sum(A,2), so that we have

```
>> sum(A,1) % which is equivalent to sum(A)
    ans = 5 7 9
>> sum(A,2) % to sum each row in (A)
    ans = 6
          15
```

Plotting Simple Functions

We start with the simplest command *plot* and use this as an opportunity to revisit the ways in which functions can be initialized. We start with initializing an array, in this case x

```
x = 0:pi/20:pi;
```

which as we know sets up a vector whose elements are (that is, a vector whose elements range from 0 to π in steps of $\pi/20$).

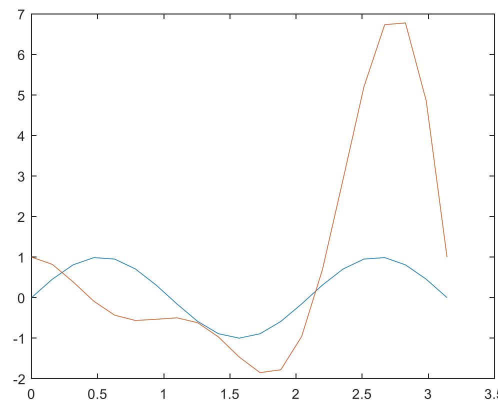
the command *size(x)* gives the array size 1x21 and the command *length(x)* (gives the maximum of the dimensions of a matrix).

We can plot simple functions, as:

plot(x, sin(x))

or more complicated examples such as:

plot(x, sin(3*x), x, x.^2.*sin(3*x)+cos(4*x))



Try these

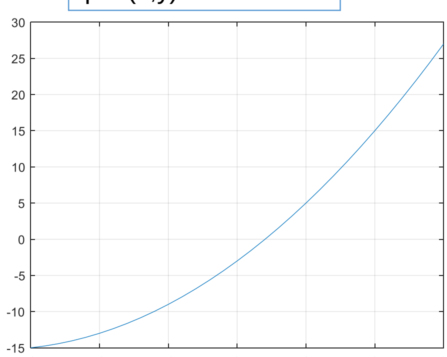
```
y = 3*x-1;
plot(x,y)
```

```
y = x.^2+3;
plot(x,y)
```

Plotting Simple Functions

Example To plot the quadratic x^2+7x-3 from x equals −3 to 3 in steps of 0.2 we use the code.

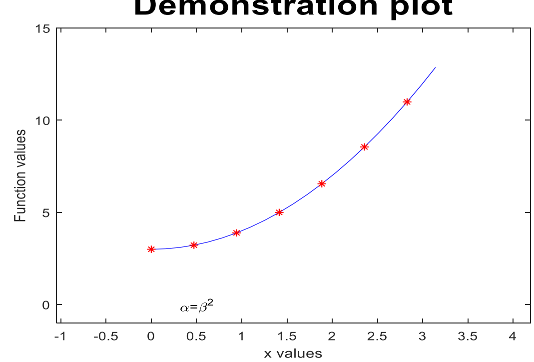
```
x = -3:0.2:3;
y = x.^2+7*x-3;
grid on
plot(x,y)
```



Example Consider the code:.

```
x = 0:pi/20:pi;
n = length(x);
r = 1:n/7:n;
y = x.^2+3;
plot(x,y,'b',x(r),y(r),'r*')
axis([-pi/3 pi+pi/3 -1 15])
xlabel('x values')
ylabel('Function values')
title('Demonstration plot','FontSize',24)
text(pi/10,0,'\alpha=\beta^2')
```

Greek letters, using the LATEX construction \alpha for α and \beta for β.

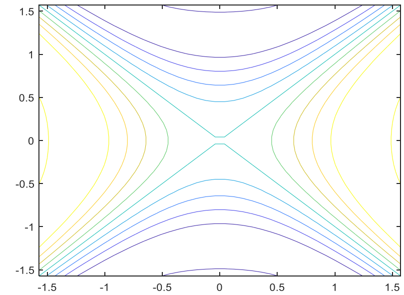


Plotting Simple Functions

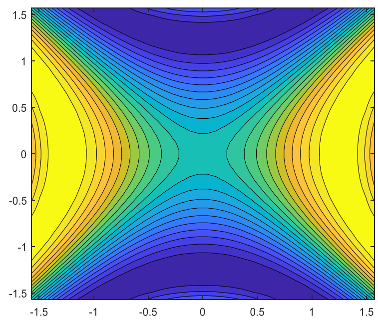
There are a wide of other plotting options available. like *loglog(x,y)* produces a log-log plot. Similarly *semilogx* and *semilogy* produces a log plot for the x and y-axis, respectively.

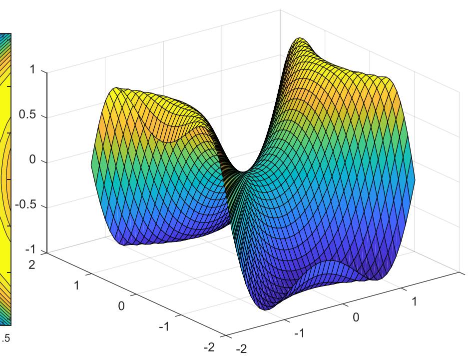
You should also be aware of the commands *clf* which clears the current figure and *hold* which holds the current figure.

The excellent features of MATLAB is the way in which it handles **two** and **three-dimensional** graphics.




```
x = linspace(-pi/2,pi/2,40);
y = x;
[X,Y] = meshgrid(x,y);
f = sin(X.^2-Y.^2);
figure(1)
contour(X,Y,f)
figure(2)
contourf(X,Y,f,20)
figure(3)
surf(X,Y,f)
```





Evaluating Polynomials and Plotting Curves

Example a code to generate the value of a specific quadratic $x^2 + x + 1$ at a specific point:

```
x = 3;
y = x^2+x+1;
disp(y)
```

In general suppose we have the general quadratic
$y = a_2x^2 + a_1x + a_0$.

The script to calculate this equation will call *quadratic.m*

Note it is also possible to obtain a complete listing of the code by typing *type quadratic* at the prompt.

There are many ways of writing polynomials, for instance this could have been written recursively as
$a_0 + x(a_1 + xa_2)$.

```
% quadratic.m
% This program evaluates a quadratic
% at a certain value of x
% The coefficients are stored in a2, a1 and a0.

str = 'Please enter the ';
a2 = input([str 'coefficient of x squared: ']);
a1 = input([str 'coefficient of x: ']);
a0 = input([str 'constant term: ']);
x = input([str 'value of x: ']);
y = a2*x*x+a1*x+a0;
% Now display the result
disp(['Polynomial value is: ' num2str(y)])
```

by typing ***help quadratic*** at the Matlab prompt to produce:

```
quadratic.m
This programme evaluates a quadratic
at a certain value of x
The coefficients are stored in a2, a1 and a0.
```

Evaluating Polynomials and Plotting Curves

Example we want to evaluate the quadratic

$y = 3x^2 + 2x + 1$. We could then use the function

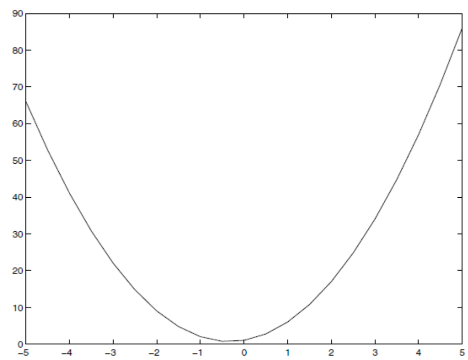
```
% evaluate_poly.m
%
function [value] = evaluate_poly(x)
value = 3*x.^2+2*x+1;
```

Now we can use the function ***evaluate_poly***, in the form evaluate_poly(2) or
x = 2; y = evaluate_poly(x).

Note: within Matlab we are able to call a function with a variety of different inputs; whether this is valid depends upon the structure of the function.
This is similar to the idea of **overloading** which is in the object orientated languages like C++ and Java.

we can now use our function to generate a vector containing the polynomial values for x vector.

```
>> x = -5:0.5:5;
>> y = evaluate_poly(x);
>> plot(x,y)
```



More on Plotting

Here we extend our plotting capability by considering the impact of a **third argument** of the plot command, such as in plot(x,y,'r.').

The colour options are

y yellow	m magenta
c cyan	r red
g green	b blue
w white	k black

the choice of **symbols** are

. point	v triangle (down)
o circle	X triangle (up)
x x-mark	< triangle (left)
+ plus	> triangle (right)
* star	p pentagram
s square	h hexagram
d diamond	

- solid
: dotted
-. dashdot
- - dashed

It is possible to control the **line style.** By drawing the line using one of the previous options:

plot more than one curve on the same graph.

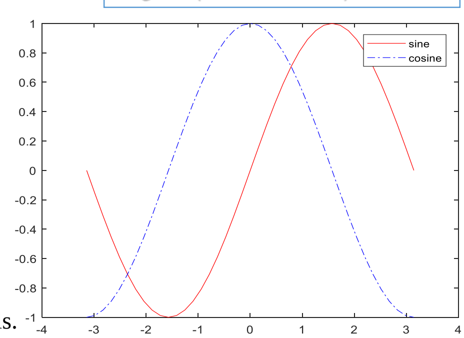
First way

```
>>x = -pi:pi/20:pi;
>>plot(x,sin(x),'r-',x,cos(x),'b-.')
```

Second way

```
x = -pi:pi/20:pi;
clf
plot(x,sin(x),'r-')
hold on
plot(x,cos(x),'b:')
hold off
legend('sine','cosine')
```

see *help legend* for details.



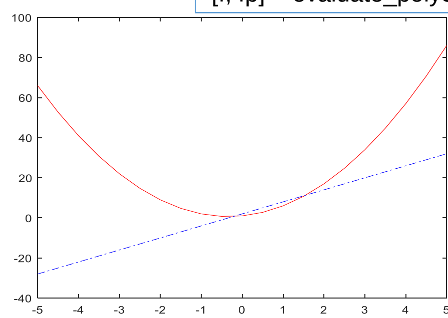
More on Plotting

we have already seen that if the function is called with a vector then the "output" is a vector..

Consider the following

```
% evaluate_poly2.m
function [f, fprime] = evaluate_poly2(x)
f = 3*x.^2+2*x+1;
fprime = 6*x+2;
```

```
x = -5:0.5:5;
[func,dfunc] = evaluate_poly2(x);
plot(x,func,'r-',x,dfunc,'b-.')
```



We can further generalize our code by passing the coefficients of the quadratic to the function, either as individual values or a vector.

```
% evaluate_poly3.m
function [f, fprime] = evaluate_poly3(a, x)
f = a(1)*x.^2+a(2)*x+a(3);
fprime = 2*a(1)*x+a(2);
```

```
x = -5:0.5:5;
a = [3 2 1];
[f, fp] = evaluate_poly3(a, x);
```

Note: ***evaluate_poly3(a,x)*** was the same as built-in function, called ***polyval***, which also evaluates polynomials.

help polyval gives more information

Functions of Functions:

- One MATLAB command which provides us with considerable freedom in writing versatile code is the command ***feval*** which loosely translates as *function evaluation.*
- *The simplest use for this command is*

```
y= feval('sin',x);
```

- evaluate the function sin at x. This is equivalent to sin(x). In general the arguments for ***feval*** are the **name of the function**, which must be either a MATLAB built-in function or a user defined function, contained between quotes, and **the value** (or values in the case of a vector) at which the function is to be evaluated.
- Consider the function $h(x) = 2\sin^2 x + 3\sin x - 1.$
- One way of writing code that would evaluate this function is

```
function [h] = fnc(x)
h = 2*sin(x).*sin(x)+3*sin(x)-1
```

- However we can recognize this function as a composition of two functions

  $g(x) = \sin(x)$ and $f(x) = 2x^2 + 3x - 1$ so that $h(x) = f(g(x))$
- We can easily write a function file *f.m* to evaluate *f(x)*

```
function [y] = f(x)
y = 2*x.^2+3*x-1;
```

Functions of Functions:

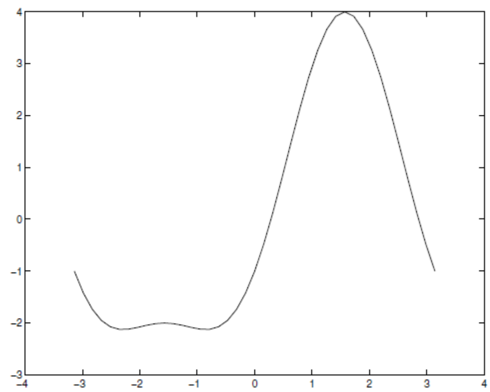
- To use this in calculating the value of the composite function *h(x) we need to be* able to pass the function name to f as an argument. We can do this with the following modification to our code

```
function [y] = f(fname,x)
z = feval(fname,x);
y = 2*z.^2+3*z-1;
```

- Calculating the function *h(x) is now as simple as*

```
x = -pi:pi/20:pi;
y = f('sin',x);
plot(x,y)
```

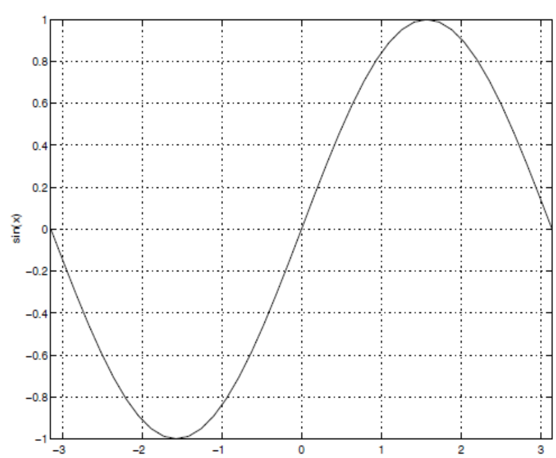
- This gives a plot of the function *h(x) as*



Functions of Functions:

A more useful example is given by the function ***plotf***

```
function plotf(fname, x)
y = feval(fname, x)
plot(x,y)
grid on
axis([min(x) max(x) min(y) max(y)])
xlabel('x')
ylabel([fname '(x)'])
```



- This function takes as input a function name and a vector x and produces a labeled plot. Notice although there are two inputs for this function there are no values output; the figure is the output.

Any Question?